Parse Tree

PROGRAM

```
<p4program> := <declarationList>

<declarationList> := { <declaration> | ; /* empty declaration */ }*

<declaration> := <variableDeclaration> | <externDeclaration> | <actionDeclaration> |
  <typeDeclaration> | <parserDeclaration> | <controlDeclaration> | <instantiation> |
  <errorDeclaration> | <matchKindDeclaration> | <functionDeclaration>

<nonTypeName> := IDENTIFIER | apply | key | actions | state | entries | type

<name> := <nonTypeName> | TYPE_IDENTIFIER

<nonTableKwName> := IDENTIFIER | TYPE_IDENTIFIER | apply | state | type

<parameterList> := { <parameter> { , <parameter> }* }

<parameter> := { <direction> } <typeRef> <name> { = <expression> }

<direction> := in | out | inout

<packageTypeDeclaration> := package <name> <optTypeParameters> ( <parameterList> )

<instantiation> := <typeRef> ( <argumentList> ) <name> ;

<optConstructorParameters> := { ( <parameterList> ) }
```

PARSER

```
<parserDeclaration> := <parserTypeDeclaration> <optConstructorParameters>
  { <parserLocalElements> <parserStates> }

<parserLocalElements> := { <parserLocalElement> }*

<parserLocalElement> := <variableDeclaration> |  <instantiation> /* | <valueSetDeclaration> */

<parserTypeDeclaration> := parser <name> <optTypeParameters> ( <parameterList> )

<parserStates> := <parserState> { <parserState> }*

<parserState> := state <name> { <parserStatements> <transitionStatement> }

<parserStatements> := { <parserStatement> }*

<parserStatement> := <assignmentOrMethodCallStatement> | <directApplication> | <parserBlockStatement> |
  <variableDeclaration> | <emptyStatement>

<parserBlockStatement> := { <parserStatements> }

<transitionStatement> := { transition <stateExpression> }

<stateExpression> := <name> ; | <selectExpression>

<selectExpression> := select ( <expressionList> ) { <selectCaseList> }

<selectCaseList> := { <selectCase> }*

<selectCase> := <keysetExpression> : <name> ;
```

```
<keysetExpression> := <tupleKeysetExpression> | <simpleKeysetExpression>

<tupleKeysetExpression> := ( <keysetExpressionList> )

<keysetExpressionList> := <simpleKeysetExpression> { , <simpleKeysetExpression }*

<simpleKeysetExpression> := <expression> /* { ( mask | range ) <expression> } */ | default | _

/* <valueSetDeclaration> := valueset '<' ( <baseType> | <tupleType> | <typeName> ) '>'
  '(' <expression> ')' <name> ';' */
```

CONTROL

```
<controlDeclaration> := <controlTypeDeclaration> <optConstructorParameters>
  { <controlLocalDeclarations> apply <blockStatement> }

<controlTypeDeclaration> := control <name> <optTypeParameters> ( <parameterList> )

<controlLocalDeclaration> := <variableDeclaration> | <actionDeclaration> | <tableDeclaration> |
  <instantiation>

<controlLocalDeclarations> := { <controlLocalDeclaration }*
```

EXTERN

```
<externDeclaration> := extern
  ( <nonTypeName> <optTypeParameters> { <methodPrototypes> } ) |
  ( <functionPrototype> ; )

<methodPrototypes> := { <methodPrototype> }*

<functionPrototype> := <typeOrVoid> <name> <optTypeParameters> ( <parameterList> )

<methodPrototype> :=
  ( <functionPrototype> ; ) |
  ( TYPE ( <parameterList> ) ; )  /* constructor */
```

TYPES

```
<typeRef> := <baseType> | <namedType> | <tupleType>

<namedType> := <typeName> | <specializedType> | <headerStackType>

<prefixedType> := { . } TYPE_IDENTIFIER

<tupleType> := tuple < <typeArgumentList> >

<headerStackType> := <typeName> [ <expression> ]

<specializedType> := <typeName> < <typeArgumentList> >

<baseType> :=
  bool | error | string | void |
  int { < <integerTypeSize> > } |
  bit { < <integerTypeSize> > } |
  varbit < <integerTypeSize> >

<integerTypeSize> := INTEGER /* | '(' <expression> ')' */

<typeOrVoid> := <typeRef> | void | IDENTIFIER  /* type variable */

<optTypeParameters> := { < <typeParameterList> > }
```

```
<typeParameterList> := <name> { , <name> }*

<realTypeArg> := _ | <typeRef>

<typeArg> := _ | <typeRef> | <nonTypeName>

<realTypeArgumentList> := <realTypeArg> { , <realTypeArg> }*

<typeArgumentList> := { <typeArg> { , <typeArg> }* }

<typeDeclaration> := <derivedTypeDeclaration> | <typedefDeclaration> | ( <parserTypeDeclaration> ; )
  | ( <controlTypeDeclaration> ; ) | ( <packageTypeDeclaration> ; )

<derivedTypeDeclaration> := <headerTypeDeclaration> | <headerUnionDeclaration> |
  <structTypeDeclaration> | <enumDeclaration>

<headerTypeDeclaration> := header <name> { <structFieldList> }

<headerUnionDeclaration> := header_union <name> { <structFieldList> }

<structTypeDeclaration> := struct <name> { <structFieldList> }

<structFieldList> := { <structField> { , <structField> }* }

<structField> := <typeRef> <name> ;

<enumDeclaration> := enum { bit < INTEGER > } <name> { <specifiedIdentifierList> }

<errorDeclaration> := error { <identifierList> }

<matchKindDeclaration> := match_kind { <identifierList> }

<identifierList> := <name> { , <name> }*

<specifiedIdentifierList> := <specifiedIdentifier> { , <specifiedIdentifier> }*

<specifiedIdentifier> := <name> { = <initializer> }

<typedefDeclaration> := ( typedef | type ) ( <typeRef> | <derivedTypeDeclaration> ) <name> ';'
```

STATEMENTS

```
<assignmentOrMethodCallStatement> := <lvalue>
  ( { < <typeArgumentList> > } ( <argumentList> ) ; ) |
  ( = <expression> ; )

<emptyStatement> := ;

<returnStatement> := return { expression } ;

<exitStatement> := exit ;

<conditionalStatement> := if ( <expression> ) <statement> { else <statement> }

/* To support direct invocation of a control or parser without instantiation. */
<directApplication> := <typeName> . apply ( <argumentList> ) ;

<statement> := <assignmentOrMethodCallStatement> | <directApplication> | <conditionalStatement> |
  <emptyStatement> | <blockStatement> | <exitStatement> | <returnStatement> | <switchStatement>

<blockStatement> := { <statementOrDeclList> }
```

```
<statementOrDeclList> := { <statementOrDeclaration> }*

<switchStatement> := switch ( <expression> ) { <switchCases> }

<switchCases> := { <switchCase> }*

<switchCase> := <switchLabel> : { <blockStatement> }

<switchLabel> := <name> | default

<statementOrDeclaration> := <variableDeclaration> | <statement> | <instantiation>
```

TABLES

```
<tableDeclaration> := table <name> { <tablePropertyList> }

<tablePropertyList> := <tableProperty> { <tableProperty> }*

<tableProperty> :=
  ( key = { <keyElementList> } ) |
  ( actions = { <actionList> } ) |
  ( const entries = { <entriesList> } ) |  /* immutable entries */
  ( { const } <nonTableKwName> = <initializer> ; )

<keyElementList> := { <keyElement> }*

<keyElement> := <expression> : <name> ;

<actionList> := { <actionRef> ; }*

<actionRef> := <prefixedNonTypeName> { ( <argumentList> ) }

<entriesList> := <entry> { <entry> }*

<entry> := <keysetExpression> : <actionRef> ;

<actionDeclaration> := action <name> ( <parameterList> ) <blockStatement>
```

VARIABLES

```
<variableDeclaration> := { const } <typeRef> <name> { = <expression> };
```

EXPRESSIONS

```
<functionDeclaration> := <functionPrototype> <blockStatement>

<argumentList> := { <argument> { , <argument> }* }

<argument> := <expression> | _

/* <kvList> := <kvPair> { ',' <kvPair> }* */

/* <kvPair> := <name> = <expression> */

<expressionList> := { <expression> { , <expression> }* }

<prefixedNonTypeName> := { . } <nonTypeName>

<lvalue> := <prefixedNonTypeName> {
    ( . <name> ) |  /* member selector */
    ( [ <indexExpression> ] )  /* array subscript */
  }*
```

```
<expression> := <expressionPrimary> { <exprOperator> <expression> }*

<expressionPrimary> := <integer> | <boolean> | <string> |
  ( { . } <nonTypeName> ) |
  ( { <expressionList> } ) |  /* <kvList> */
  ( ( <expression> ) ) |
  ( ( ! | ~ | - ) <expression> ) |  /* unary expression */
  ( <namedType> | error ) |  /* member selector, function call */
  ( ( <typeRef> ) <expression> )  /* cast */

<exprOperator> := <binaryOperator> |
  ( . <name> ) |  /* member selector */
  ( [ <indexExpression> ] ) |  /* array subscript */
  ( ( <argumentList> ) ) |  /* function call */
  ( < <realTypeArgumentList> > ) |
  ( = <expression> )  /* <kvPair> */

<indexExpression> := <expression> { : <expression> }

<integer> := INTEGER

<boolean> := true | false

<string> := STRING

<binaryOperator> := * | / | + | - | <= | >= | < | > | != | == | || | && | | | & | << | >>
```

Syntax Tree

PROGRAM

1. <p4program> := <declarationList>$_{\text{decl\_list}}$

2. <declarationList> := { <declaration>$_{[0..n]}$ }*

3. <declaration> := (<variableDeclaration> | <externDeclaration> | <actionDeclaration> | <functionDeclaration> | <parserDeclaration> | <parserTypeDeclaration> | <controlDeclaration> | <controlTypeDeclaration> | <typeDeclaration> | <errorDeclaration> | <matchKindDeclaration> | <instantiation>)$_{\text{decl}}$

4. <name> := STRING$_{\text{strname}}$

5. <parameterList> := { <parameter>$_{[0..n]}$ }*

6. <parameter> := { **in** | **out** | **inout** }$_{\text{direction}}$ <typeRef>$_{\text{type}}$ <name>$_{\text{name}}$ { <expression> }$_{\text{init\_expr}}$

7. <packageTypeDeclaration> := <name>$_{\text{name}}$ { <typeParameterList> }$_{\text{type\_params}}$ <parameterList>$_{\text{params}}$

8. <instantiation> := <typeRef>$_{\text{type\_ref}}$ <argumentList>$_{\text{args}}$ <name>$_{\text{name}}$

PARSER

1. <parserDeclaration> := <typeDeclaration>$_{\text{proto}}$ { <parameterList> }$_{\text{ctor\_params}}$ <parserLocalElements>$_{\text{local\_elements}}$ <parserStates>$_{\text{states}}$

2. <parserTypeDeclaration> := <name>$_{\text{name}}$ { <typeParameterList> }$_{\text{type\_params}}$ <parameterList>$_{\text{params}}$

3. <parserLocalElements> := { <parserLocalElement>$_{[0..n]}$ }*

4. <parserLocalElement> := (<variableDeclaration> | <instantiation>)$_{\text{element}}$

5. <parserStates> := { <parserState>$_{\text{[0..n]}}$ }+

6. <parserState> := <name>$_{\text{name}}$ <parserStatements>$_{\text{stmt\_list}}$ <transitionStatement>$_{\text{transition\_stmt}}$

7. <parserStatements> := { <parserStatement>$_{\text{[0..n]>}}$ }*

8. <parserStatement> := (<assignmentStatement> | <functionCall> | <directApplication> | <parserBlockStatement> | <variableDeclaration>)$_{\text{stmt}}$

9. <parserBlockStatement> := <parserStatements>$_{\text{stmt\_list}}$

10. <transitionStatement> := <stateExpression>$_{\text{stmt}}$

11. <stateExpression> := (<name> | <selectExpression>)$_{\text{expr}}$

12. <selectExpression> := <expressionList>$_{\text{expr\_list}}$ <selectCaseList>$_{\text{case\_list}}$

13. <selectCaseList> := { <selectCase>$_{\text{[0..n]}}$ }*

14. <selectCase> := <keysetExpression>$_{\text{keyset\_expr}}$ <name>$_{\text{name}}$

15. <keysetExpression> := (<tupleKeysetExpression> | <expression> | <defaultKeysetExpression> | <dontcareKeysetExpression>)$_{\text{expr}}$

16. <tupleKeysetExpression> := <keysetExpressionList>$_{\text{expr\_list}}$

17. <keysetExpressionList> := { <simpleKeysetExpression>$_{\text{[0..n]}}$ }+

CONTROL

1. <controlDeclaration> := <typeDeclaration>$_{\text{proto}}$ { <parameterList> }$_{\text{ctor\_params}}$ <controlLocalDeclarations>$_{\text{local\_decls}}$ <blockStatement>$_{\text{apply\_stmt}}$

2. <controlTypeDeclaration> := <name>$_{\text{name}}$ { <typeParameterList> }$_{\text{type\_params}}$ <parameterList>$_{\text{params}}$

3. <controlLocalDeclarations> := { <controlLocalDeclaration>$_{\text{[0..n]}}$ }*

4. <controlLocalDeclaration> := (<variableDeclaration> | <actionDeclaration> | <tableDeclaration> | <instantiation>)$_{\text{decl}}$

EXTERN

1. <externDeclaration> := (<externTypeDeclaration> | <functionPrototype>)$_{\text{decl}}$

2. <externTypeDeclaration> := <name>$_{\text{name}}$ { <typeParameterList>$_{\text{type\_params}}$ } <methodPrototypes>$_{\text{method\_protos}}$

3. <methodPrototypes> := { <methodPrototype>$_{\text{[0..n]}}$ }*

4. <functionPrototype> := { <typeRef> }$_{\text{return\_type}}$ <name>$_{\text{name}}$ { <typeParameterList> }$_{\text{type\_params}}$ <parameterList>$_{\text{params}}$

TYPES

1. <typeRef> := (<baseTypeBool> | <baseTypeError> | <baseTypeInteger> | <baseTypeBit> | <baseTypeVarbit> | <baseTypeString> | <baseTypeVoid> | <namedType> | <tupleType>)$_{\text{type}}$

2. <namedType> := (<name> | <specializedType> | <headerStackType>)$_{\text{type}}$

3. <tupleType> := <typeArgumentList>$_{\text{type\_args}}$

4. <headerStackType> := <name>$_{\text{name}}$ <expression>$_{\text{stack\_expr}}$

5. <specializedType> := <name>$_{\text{name}}$ <typeArgumentList>$_{\text{type\_args}}$

6. <baseTypeBool> := <name>$_{\text{name}}$
7. <baseTypeInteger> := <name>$_{\text{name}}$ { <integerTypeSize> }$_{\text{size}}$
8. <baseTypeBit> := <name>$_{\text{name}}$ { <integerTypeSize> }$_{\text{size}}$
9. <baseTypeVarbit> := <name>$_{\text{name}}$ <integerTypeSize>$_{\text{size}}$
10. <baseTypeString> := <name>$_{\text{name}}$
11. <baseTypeVoid> := <name>$_{\text{name}}$
12. <baseTypeError> := <name>$_{\text{name}}$

13. <integerTypeSize> := INTEGER$_{\text{size}}$

14. <typeParameterList> := { <name>$_{[0..n]}$ }+

15. <realTypeArg> := (_ | <typeRef>)$_{\text{arg}}$

16. <typeArg> := (_ | <typeRef> | <name>)$_{\text{arg}}$

17. <realTypeArgumentList> := { <realTypeArg>$_{[0..n]}$ }+

18. <typeArgumentList> := { <typeArg>$_{[0..n]}$ }*

19. <typeDeclaration> := (<derivedTypeDeclaration> | <typedefDeclaration> | <parserTypeDeclaration> | <controlTypeDeclaration> | <packageTypeDeclaration>)$_{\text{decl}}$

20. <derivedTypeDeclaration> := (<headerTypeDeclaration> | <headerUnionDeclaration> | <structTypeDeclaration> | <enumDeclaration>)$_{\text{decl}}$

21. <headerTypeDeclaration> := <name>$_{\text{name}}$ <structFieldList>$_{\text{fields}}$

22. <headerUnionDeclaration> := <name>$_{\text{name}}$ <structFieldList>$_{\text{fields}}$

23. <structTypeDeclaration> := <name>$_{\text{name}}$ <structFieldList>$_{\text{fields}}$

24. <structFieldList> := { <structField>$_{[0..n]}$ }*

25. <structField> := <typeRef>$_{\text{type}}$ <name>$_{\text{name}}$

26. <enumDeclaration> := INTEGER$_{\text{type\_size}}$ <name>$_{\text{name}}$ <specifiedIdentifierList>$_{\text{fields}}$

27. <errorDeclaration> := <identifierList>$_{\text{fields}}$

28. <matchKindDeclaration> := <identifierList>$_{\text{fields}}$

29. <identifierList> := { <name>$_{[0..n]}$ }+

30. <specifiedIdentifierList> := { <specifiedIdentifier>$_{[0..n]}$ }+

31. <specifiedIdentifier> := <name>$_{\text{name}}$ { <initializer> }$_{\text{init\_expr}}$

32. <typedefDeclaration> := (<typeRef> | <derivedTypeDeclaration>)$_{\text{type\_ref}}$ <name>$_{\text{name}}$

STATEMENTS

1. <assignmentStatement> := <lvalueExpression>$_{\text{lhs\_expr}}$ <expression>$_{\text{rhs\_expr}}$

2. <functionCall> := (<expression> | <lvalueExpression>)$_{\text{lhs\_expr}}$ <argumentList>$_{\text{args}}$

3. <returnStatement> := { <expression> }$_{\text{expr}}$

4. <exitStatement>

5. <conditionalStatement> := <expression>$_{\text{cond\_expr}}$ <statement>$_{\text{stmt}}$ <statement>$_{\text{else\_stmt}}$

6. <directApplication> := <typeName>$_{\text{name}}$ <argumentList>$_{\text{args}}$

7. <statement> := (<assignmentStatement> | <functionCall> | <directApplication> | <conditionalStatement> | <emptyStatement> | <blockStatement> | <exitStatement> | <returnStatement> | <switchStatement>)$_{\text{stmt}}$

8. <blockStatement> := <statementOrDeclList>$_{\text{stmt\_list}}$

9. <statementOrDeclList> := { <statementOrDeclaration>$_{[0..n]}$ }*

10. <switchStatement> := <expression>$_{\text{expr}}$ <switchCases>$_{\text{switch\_cases}}$

11. <switchCases> := { <switchCase>$_{[0..n]}$ }*

12. <switchCase> := <switchLabel>$_{\text{label}}$ <blockStatement>$_{\text{stmt}}$

13. <switchLabel> := (<name> | **default**)$_{\text{label}}$

14. <statementOrDeclaration> := (<variableDeclaration> | <statement> | <instantiation>)$_{\text{stmt\_or\_decl}}$

TABLES

1. <tableDeclaration> := <name>$_{\text{name}}$ <tablePropertyList>$_{\text{prop\_list}}$

2. <tablePropertyList> := { tableProperty$_{[0..n]}$ }+

3. <tableProperty> := (<keyProperty> | <actionsProperty> | <entriesProperty> | <simplePropery>)$_{\text{prop}}$

4. <keyProperty> := <keyElementList>$_{\text{keyelem\_list}}$

5. <keyElementList> := { <keyElement>$_{[0..n]}$ }*

6. <keyElement> := <expression>$_{\text{expr}}$ <name>$_{\text{match}}$

7. <actionsProperty> := <actionsList>$_{\text{action\_list}}$

8. <actionList> := { <actionRef>$_{[0..n]}$ }*

9. <actionRef> := <name>$_{\text{name}}$ <argumentList>$_{\text{args}}$

10. <entriesProperty> := <entriesList>$_{\text{entries\_list}}$

11. <entriesList> := { <entry>$_{[0..n]}$ }+

12. <entry> := <keysetExpression>$_{\text{keyset}}$ <actionRef>$_{\text{action}}$

13. <simplePropery> := <name>$_{\text{name}}$ <expression>$_{\text{init\_expr}}$

14. <actionDeclaration> := <name>$_{\text{name}}$ <parameterList>$_{\text{params}}$ <blockStatement>$_{\text{stmt}}$

VARIABLES

1. <variableDeclaration> := <typeRef>$_{\text{type}}$ <name>$_{\text{name}}$ { <expression> }$_{\text{init\_expr}}$

EXPRESSIONS

1. <functionDeclaration> := <functionPrototype>$_{\text{proto}}$ <blockStatement>$_{\text{stmt}}$

2. <argumentList> := { <argument>$_{[0..n]}$ }*

3. <argument> := (<expression> | _)$_{\text{arg}}$

4. <kvPair> := <name>$_{\text{name}}$ <expression>$_{\text{init\_expr}}$

5. <expressionList> := { <expression>$_{[0..n]}$ }*

6. <lvalueExpression> := (<name> | <memberSelector> | <arraySubscript>)$_{\text{expr}}$

7. <expression> := (<integerLiteral>| <booleanLiteral> | <stringLiteral> | <name> | <namedType> | <expressionList> | castExpression | <unaryExpression> | <binaryExpression> | <memberSelector> | <arraySubscript> | <functionCall> | <kvPair>)$_{\text{expr}}$ { <realTypeArgumentList> }$_{\text{type\_args}}$

8. <castExpression> := <typeRef>$_{\text{type}}$ <expression>$_{\text{expr}}$

9. <unaryExpression> := OPERATOR$_{\text{op}}$ <expression>$_{\text{operand}}$

10. <binaryExpression> := <expression>$_{\text{left\_operand}}$ OPERATOR$_{\text{op}}$ <expression>$_{\text{right\_operand}}$

11. <memberSelector> := <expression>$_{\text{lhs\_expr}}$ <name>$_{\text{name}}$

12. <arraySubscript> := <expression>$_{\text{lhs\_expr}}$ <indexExpression>$_{\text{index\_expr}}$

13. <indexExpression> := <expression>$_{\text{start\_index}}$ { <expression> }$_{\text{end\_index}}$

14. <integerLiteral> := INTEGER$_{\text{value}}$ INTEGER$_{\text{width}}$

15. <booleanLiteral> := INTEGER$_{\text{value}}$

16. <stringLiteral> := STRING$_{\text{value}}$